In the Specification:

Please amend the Specification as shown below. Applicant respectfully submits that the proposed amendments correct informalities in the Specification and that no new matter is being added.

Please add new paragraph [0013.1] as shown below

[0013.1] The terms JAVA, JAVA 2 Enterprise Edition (J2EE), JAVA Message Service (JMS), Enterprise JAVA Bean (EJB), and JAVA Management Extensions (JMX), are trademarks of Sun Microsystems, Inc. The terms WebLogic, and WebLogic Server, are trademarks of BEA Systems, Inc.

Please replace paragraph [0002] with that shown below:

[0002] This application is related to co-pending U.S. Utility Patent Application Serial No. [[_____]] 10/602,037 by Kathiravan Sengodan entitled "SYSTEM AND METHOD FOR JAVA MESSAGE SERVICE MARK-UP LANGUAGE", filed [[_____]] June 23, 2003 (BEAS-01351US1), U.S. Utility Patent Application Serial No. [[_____]] 10/601,898 by Kathiravan Sengodan entitled "WEB-BASED INTERFACE FOR JAVA MESSAGE SERVICE MARK-UP LANGUAGE", filed[[_____]] June 23, 2003 (BEAS-01351US2), and U.S. Utility Patent Application Serial No. [[_____]] 10/601,929 by Kathiravan Sengodan entitled "COMMAND-LINE INTERFACE SYSTEM AND METHOD FOR JAVA MESSAGE SERVICE MARK-UP LANGUAGE", filed [[_____]] June 23, 2003 (BEAS-01351US3), which applications are also incorporated herein by reference.

Please replace paragraph [0013] with that shown below:

[0013] In accordance with one embodiment the invention comprises a system having an extensible engine that performs the task of parsing input data and converting it to [[Java]] JAVA

JMS/JMX API, and then executes the JMSML program. In other embodiments the invention includes methods of using JMSML to access JMS/JMX APIs.

Please replace paragraph [0033] with that shown below:

[0033] An embodiment of the invention provides a markup language (referred to herein as JMSML), and user interfaces, that are useful for creating simple and reusable JMS components. The JMS components can be used for performing JMX operations necessary for dynamic configuration, management and runtime monitoring of a JMS server, in addition to performing enterprise messaging using the JMS operations. JMSML is a mark-up language designed and developed to make [[Java]] JAVA Message Service (JMS) and [[Java]] JAVA Management Extensions (JMX) programming easy by hiding the JMS and JMX [[Java]] JAVA API complexity behind a few easy-to-use XML tags. Embodiments of the invention include systems and methods that utilize JMSML to abstract the complexity of the JMS and JMX APIs into a relatively small plurality of tags.

Please replace paragraph [0034] with that shown below:

[0034] In accordance with one embodiment the invention comprises a system having an extensible engine that performs the task of parsing input data and converting it to [[Java]] JAVA JMS/JMX API, and then executes the JMSML program. In other embodiments the invention includes methods of using JMSML to access JMS/JMX APIs.

Please replace paragraph [0035] with that shown below:

[0035] A typical JMS application development process involves configuring and managing the JMS server components, such as the JMSConnectionFactory, JMSDestination, etc., on the JMS Provider application server, and then writing JMS application clients that will make use of these administered server-side objects to perform [[Java]] JAVA messaging. JMS application

clients are written in Java using the JMS API, and are categorized into two types of programs: Producers, and Consumers. Producers create various JMS Message types (like Text, XML, Object, Stream, Bytes) and send them to the JMS destinations (Queues and Topics). Consumers receive the messages from the JMS destinations (Queues and Topics), both synchronously and asynchronously. The JMS API enables both producers and consumers to utilize various qualities of service (QOS) that are provided by the underlying JMS implementation and by the JMS providers (such as transaction, acknowledgment, etc.).

Please replace paragraph [0039] with that shown below:

[0039] In accordance with one embodiment, JMSML is implemented as a number of neatly abstracted [[Java]] JAVA components grouped together as the extensible JMSML engine., which utilizes technologies like JMS, JMX and XML to deliver the power that a JMS application developer needs, while hiding all the finer details of [[Java]] JAVA and JMS API behind a few XML tags.

Please replace paragraph [0108] with that shown below:

[0108] The JMSML language can be supported in a variety of different environments and through a variety of different interfaces, which can be presently summarized as follows:

1. JMSML Integrated Development Environment (IDE);
2. Standalone Java Client; or,
3. [[Java]] JAVA Servlet webapp.

Please replace paragraph [0132] with that shown below:

[0132] **Figure 16** shows a flowchart of the operation of a JMSML system that includes a command-line interface in accordance with an embodiment of the invention. In step **530**, the user opens a command-line interface at the client. The user then, in step **532**, enters commands to invoke a [[Java]] JAVA method. In step **534**, the corresponding command (in XML format) is

passed to the JMS engine. In step **536**, the JMSML engine generates the appropriate JMS/JMX commands and communicates them to the relevant server. Results may be optionally generated, and in step **538** stored in an optional results file.

Please replace paragraph [0150] with that shown below:

[0150] The next time the same command line is executed, JMSML then treats the "jmstest.xml" as a test case and prints out the test pass/fail result to the stdout. As can be seen, without writing a [[Java]] JAVA program, we have quickly written a complete test case for BEA WebLogic JMS and JMX features.